***Отчет по Task2***

**Первый этап** — подготовка данных. Все значение были считаны из *.txt* файла и переделаны в *Dataframe* для дальнейшего удобства. Значения предикторов, по всей видимости, нормированы в промежутке *[0, 1]*, а прогнозируемая величина принимает целочисленные значения. Изучив детальнее все данные, увидел, что y — не только целочисленная переменная, но и значения, принимаемые ею — ограничены. Из чего можно сделать вывод, что задача может быть отнесена к классификации, а не регрессии. Также оказаллось, что относительно сильная корреляция есть только между y и предикторами *x\_1, x\_2, x\_4, x\_5*, с остальными же переменными корреляция не только слабая, но и обратная. Поэтому возмем обратное отношение этих предикторов, чтобы все корреляции были прямыми. Очевидно, что после данной операции значения соотвествующих переменных будут уже больше 1, поэтому, возможно, стоит отнормировать эти значения с помощью *MinMaxScalar*. После анализа окончателных данных понял, что даже после корректировки некоторых предикторов ситуация с корреляцией особо не поменялась, поэтому эта задача точно относится к классфикации, и далее нужно исследовать различные алгоритмы для соотвествующей задачи. Чтобы оценить качество набора данных, изучим его сбалансированность. Распредение данны по классам оказалось весьма неравномерным, для устранения этой особенностиы будем использовать готовые решения (в виде параметров алгоритмов классификации, например *class\_weights*).

**Второй этап** — тестирование различных алгоритмов классификации. Все предикторы лежат в диапозоне от 0 до 1, поэтому, предположу, необязательно производить стандартизацию и масштабирование этих переменных. Соотвественно, не нужно строить pipeline, а можно сразу приступать к созданию класса модели и дальнейшего подбора параметров путем прохода по сетке гиперпараметров. В ходе выполнения задания протестировал 4 алгоритма: *K Nearest Neighboors, SVM, Multy-Layer Perceptron* и Байесовский классификатор. Отмечу, что в актуальной версии программы ф-ия, использующая *MLP*, никак не связана с *MLP Classifier*, который я тестировал в самом начале. К сожалению, все алгоритмы показали плохой результат — метрика accuracy (хорошо подходящия для многоклассовой классификации) в лучше случае достигала 18-20%. Можно, конечно, было вычислять *confusion matrix*, а затем по ней считать, например, *ROC AUC или precision/recall* (делая по сути пары «класс i» и «все остальные классы, кроме i», то есть сводя к бинарной классификации), но ситуация от этого не изменилась бы. В данных явно выражено отсутсвие баланса, есть класс с 1 представителем, а есть класс с 80-90 представителями, поэтому алгоритмы, обученные на таком датасете, обладают скудной обобщающей способностью. Однако удалось оценить значимость каждого предиктора, в среднем можно было обросить *x\_7, x\_9, x\_10*, а значимыми оказались предикторы с первого по четвертый. Поэтому было принято решение кардинально изменить подход.

**Третий этап** — использование регрессии для классификации (?!). Посмотрев на статистику по переменным (.html файл), а особенно на игрек, заметил, что его распределение близко к стандартному распределению. Если сводить на язык статистики, то в голове появилась такая мысль: *y ~ N(w^T\*X, sigma^2)*, где *w* — веса, *X* — вектор предикторов, а сигма — дисперсия. То есть чтобы получать прогноз y, на нужно было бы как-то вычислять ф-ию плотности распределения, а потом итегрировать ее по допустимому множесту. Также для оценки параметров распределения скорее всего потреболалось бы использовать метод максимального правдоподобия. Задача получается, если можно так выразиться, *overkill*, поэтому итоговый метод следующий. На неравномерной выборке сложно получить хорошую обобщающую способность, поэтому мне голову пришла идея интерполировать *y = f(x\_1, …, x\_10)* 6а имеющемся наборе данных. Имея распределенный особым образом игрек, а также полное отсутствие корреляции (см. первый этап), будем сразу использовать *MLP Regressor*, так как нейронные сети даже с ф-ией активации relu хорошо подходят для построения нелинейных ф-ий. После обучения этот алгортм по входным предикторам выдает непрерывную величину, поэтому, чтобы получить номер класса, округляем эту величину по правилам математики. Таким образом, непрерывная ф-ия, которую построил *MLP*, становится ступенчатой, что уже подходит для изначальной задачи. Так как нужно оценить качество классификации с помощью такого алгоритма, то регрессионные метрики будут абсолютно неиформативны, а использовать метрику классификации не позволяет *GridSearchCV*. Не беда - напишем свою. Спрогнозированные величины она округляет, как описал выше, и уже их, как и реальные ответы, передает метрике acuuracy (см. в коде ф-ию *func\_reg\_class\_error()*). Таким образом, метрика классификации получилась обернута в метрику регрессии. Опять же, вполне возможно, что можно было использовать *confusion matrix* и *ROC AUC*, как я описывал выше, но, вероятно, на результат это сильно не повлияло бы. Резюмируя, самописная метрика оказалась весьма репрезентативной, и уже с ней можно было подбирать оптимальные параметры модели. В конце концов, точность всего этого алгоритма составляет 60-63%, что, конечно же, намного лучше, чем 18-20, но все равно не могу сказать, что результат феноменальный в смысле одной лишь метрики. Но, учитывая набор данных и вообще сложную ситуацию, могу сказать, что модель показывается себя весьма хорошо. Для удобства пользователя, последняя обученная модель с метрикой около 61% или 63% сохранена файл, который можно загрузить и сразу использовать, то есть подавать на вход предикторы и получать прогнозы без всякого обучения (см. последние строки кода).